ProKEt - Documentation
25/01/2011

Martina Freiberg

12. April 2011

Inhaltsverzeichnis

1. General

1.1. Ingredients of ProKEt dialogs

To provide an overview, the 'ingredients' of ProKEt dialogs are listed in the following. Those are all file types and mechanisms needed to assemble a final dialog.

- A **specification** of the dialog
  - The complete XML dialog specification (pure prototyping)
  - A d3web knowledge base and a minimal XML dialog specification (d3web/extensible prototyping)
- A **dialog Servlet**
  - `de.d3web.proket.run.DialogServlet` (pure prototyping)
  - `de.d3web.proket.d3web.run.D3webDialogServlet` (d3web/extensible prototyping)

  `DialogServlet` and `D3webDialogServlet` are the basic development servlets for running pure or extensible prototypes, respectively. For creating own / tailored servlets (that may, e.g., be project-specific, or for calling other rendering mechanisms or the like), see Section 2.12.
  Those servlets can either be called separately, but dialogs specified according to Sections 2.5 and 2.6 are also automatically added to the `ControlCenter` Servlet, that provides a listing of all currently active and ready-to-use dialogs and prototypes in ProKEt.
- **Appropriate renderers** that are called/initiated by the servlets first, and may further call themselves or required sub-renderers recursively.
  An overview of already existing renderer classes both for pure and d3web prototyping is listed in Appendix C
- **Appropriate StringTemplate files** that specify the 'widgets' of the dialog, i.e., all needed HTML elements and which are filled within the renderers
- **JavaScript files** for providing interactivity. For pure prototype dialogs, at least the `code.js` file is required, for d3web dialogs at least `d3webBasic.js` and `d3web.js` are required. All those JS files are found in the `src/main/webapps/js` folder.
  If further interactivity/functionality is needed, either those files need to be extended, or additional files are to be provided and included.
- Some **CSS** for styling

1.2. Usage / Use Cases

For ProKEt, there are mainly two use cases:

- Pure Prototyping
- Prototyping with knowledge base integration

Regarding *Pure Prototyping*, the main goal is to enable quick experiments with a more informal type of knowledge (data) base in the form of an XML file (see Section 2.5). Thereby, different dialog types, ßtructuringöf the knowledge, as well as general style aspects, can be investigated in an quick & easy manner. The complete workflow is described in Section 2.5.

Prototyping with knowledge base integration on the other hand provides the possibilities:

- to test knowledge base - prototype-type combinations
  (e.g., in case a knowledge base already exists or is specified due to the project, or when certain constraints regarding the prototype/dialog type are already given due to the project)
- to use the knowledge base as a specification of the desired interactivity (follow-up questions, abstractions, etc.)

Therefore, a d3web knowledge base can be loaded and used with the tool as a "database", and thus in that regards replaces the XML specification). Regarding the definition of specific properties—e.g. column-style for questionnaire, or properties of a single question only—a tailored 'minimal XML' specification is still needed. The complete workflow is described in Section 2.6.

1.3. Requirements

ProKEt first and basically is a Java-Project, that is best used from within an IDE as, e.g, Eclipse. The program logic is implemented in Java, as is also is the d3web toolkit. The resulting prototypes are based on HTML, StringTemplate, JavaScript/AJAX, CSS, and are assembled and deployed as Servlets. Essential requirements are:

- Java installation — currently used: Version Java 1.6.0_13
  http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Tomcat (or other Servlet-enabling server) — currently used: Apache Tomcat 6.0.28
  http://tomcat.apache.org/download-60.cgi
- Maven (for project & dependency management) — currently used: Apache Maven 2.2.1
  http://maven.apache.org/download.html

- SVN (standalone console-based or Eclipse/IDE SVN integration for repository access)
- Eclipse (or similar) — currently used: **Eclipse Helios for Java EE (important!) users** Release 1, 2010
  http://www.eclipse.org/downloads/

  Specific additional requirements for Eclipse which can be loaded via the Help–Install New Software... Dialog (SVN stuff only if integrated SVN is desired):
  - Maven Integration for Eclipse — currently: 0.10.2
    http://m2eclipse.sonatype.org/sites/m2e
  - Maven Integration for WTP (important for web project setup) — currently: 0.11.1
    http://m2eclipse.sonatype.org/sites/m2e-extras
  - Subversive Integration for M2Eclipse — currently 1.0.2
    http://www.polarion.org/projects/subversive/download/integrations/update-site/
  - Subversive SVN Team Provider — currently 0.7.9
    http://download.eclipse.org/releases/helios (unter COLLABORATION)
  - SVNKit 1.3.2. or Native JavaHL 1.6 (SVN connectors)
    http://community.polarion.com/projects/subversive/download/eclipse/2.0/helios-site/
- Text/XML/Web Editors—optionally, if not integrated in IDE or others are desired

2. HowTo...

2.1. ...general Setup—IDE etc

Required projects (and links for installation files) are listed in Section 1.3. Install Java and Tomcat. Unpack Maven somewhere on your disc. Install Eclipse with all the listed plugins/features. Set it up for using the installed Tomcat, Java, and the unpacked Maven installation (NOT the Eclipse integrated one!), as well as SVN for repository access.

2.2. ...get and set up required projects in the workspace

- **(complete) d3web project**:
  https://isci.informatik.uni-wuerzburg.de/svn/d3web-core → trunk → d3web
- **d3web-ProKEt project**:
  https://isci.informatik.uni-wuerzburg.de/svn/Research → trunk → Research
- **Checkout d3web and d3web-ProKEt as Maven projects**
- **d3web-ProKEt Project properties: adapt deployment**
  maven dependencies need to be transferred into Web-INF/lib:
  right click project → Deployment Assembly → Add... → Java Build Path Entries → Maven Dependencies
- **Maven package & install d3web project**
  (right click: Maven → update snapshots, dependencies, project configuration)
- **Maven install ProKEt**
  (right click: Maven → update snapshots, dependencies, project configuration)
- **Run ControlCenter (or any other built single Dialog Servlet) on server from within Eclipse**:
  Select/Open ControlCenter.java → Run as... (from run menu) → Run on server

2.3. ...get familiar with ProKEt (data- and folder structures)

In the following, a listing of the data- and folder structures of the ProKEt project, in the hierarchical order as it also appears in the project itself.

d3web-ProKEt

- **Deployment Descriptor: d3webProKEt**
  is the web.xml file, specifies context stuff, references, servlets etc
- **Java Resources**
  - **src/main/java**
    Java source code (see below for specific Java package structures)
  - **src/main/resources** Other than java resources
    - **archive**Storage for old / working / todo / project specific stuff
      - **common** stuff common to several projects, e.g. ProKEt documentation, wiki pages (kb def),...
      - **css** archived style files (working)
      - **kb** archived knowledge bases (working)
      - **projects** archived project stuff, should contain subfolder for each project, and within there shoud store at least the needed dialog specs files, and necessary custom css if used.
      - **specs** archived dialog specifications, subfolders d3web and (pure) prototypes
      - **stringtemp** archived HTML/StringTemplate files
      - **toCheck** Stuff that is currently archived and not used, yet also unclear whether it works. Needs to be checked.
    - **controlcenterResources** Some stuff for the ControlCenter or single dialog webapps templates (currently without subfolders)
    - **specs** Specification of prototypes
      - **d3web** Here go the d3web knowledge bases and corresponding specification XMLs
      - **prototypes** Here go pure-prototype specification XMLs
    - **stringtemp** StringTemplate and CSS files
      - **css** CSS style files. Saved as StringTemplate files (ending .st) also, as to ensure the usage of variables also in CSS styles.
      - **html** StringTemplate files specifying the HTML framework for dialogs and dialog elements (questions, answers, tooltops, etc...).
  - **src/test/java**
    for tests, not in use currently :-/
  - **src/test/resources**
    for tests, not in use currently :-/
  - **Libraries** References/navigates to (automatically) linked libraries as Tomcat, EAR, JRE, WebApp Libraries and Maven Dependencies (as specified in the pom.xml)
- **JavaScript Resources** JavaScript references and navigator
  - **src/main/webapp** everthing JS, as contained in the webapp folders
  - **ECMAScript Built-In** and **ECMA 3 Browser Support**
- **.settings** Contains all necessary .settings files. As an exceptions, those settings do NOT need to be excluded from synchronizing (as in other projects) as they are necessary to get the Maven + WTP project config to work
- **src** The sources
  - **main** Non-test sources
    - **java** contains above src/main/java
    - **resources** contains above src/main/resources
    - **webapp** Additional stuff needed for the webapp
      - **cases** Storage for persistence. Here the xml files containing the data are stored.
      - **img** Images needed for the webapp, e.g. global icons (for tooltip sign etc)

        - **js** Used JavaScript: custom JavaScript files needed for pure and d3web integrating prototyping
        - **libsExternal** External JavaScript libraries: jquery and jqueryUi stuff in corresponding subfolders
        - **META-INF**
        - **WEB-INF** Contains web.xml that is also referenced by Deployment Descriptor
        - **index.jsp** Optional index file, starting point of the webapp
          redirects to ControlCenter page per default
    - **test** Test sources, not used currently
- **target** Resulting .war file to be put on server, containing everything compiled into what is required to run either the ControlCenter webapp (more than one dialog at one time) or single Dialog Servlets.
- **.classpath** Classpath configuration
- **.project Project specific settings**
- **pom.xml** Specification of Maven dependencies (dependent projects etc)
- **TODO.txt** mostly "global"ToDos, i.e., required for the entire ProKEt project. ToDos for specific dialog projects (e.g., Hernia) should be stored in the archive/projects/project_folder location

2.4. ...run dialogs with ProKEt (basic workflow)

ProKEt—with a basic set of renderers, specifications, templates, styles for several basic dialog types etc—provides an easy mechanism to create new (slightly adapted) dialogs.
A listing of currently existing (and working) dialog types—both pure prototype and d3web-based dialogs—is provided in Appendix D (with screenshots).

A new dialog, that may slightly extend/adapt existing dialog types (say, with regards to the applied CSS), is created by:

- Creating the specification (see Sections 2.5)—if desired with slight adaptions, e.g., referencing other CSS styles, creating other structuring of elements by using questionnaires, etc.
- Placing the specification(s) in the corresponding, appropriate specification folder `src/main/resources/specs/prototypes` or `src/main/resources/specs/d3web`
- Compiling the d3web-ProKEt project: maven 'clean package install', with 'update snapshots'
- Refreshing the target folder
- Running the `ControlCenter` servlet in your browser by calling
  `http://localhost:8080/d3web-ProKEt/ControlCenter`
  (the specific localhost link might vary depending on the respective local tomcat installation/configuration)

- Selecting the dialog in the respective sub-menu (Pure Prototype Dialogs / d3web Dialogs)
  Dialogs are listed with the name of their specification file, ignoring the file ending. If a dialog has, for example, been specified as `dialog1.xml` as pure prototype, it is listed in the Pure Prototype Dialogs sub-menu as `dialog1`

2.5. ...create a pure prototype (pure XML specification)

This HowTo explains, how to create a prototype purely with the XML specification (e.g., for quick mockups) and assuming that existing renderers / css styles / StringTemplate files are used.
(→ adapting styles, StringTemplate templates, and renderers: see 2.7, 2.8, and 2.9)

Definition of all 'knowledge' objects (questions, answers) and their properties (question types etc) is done in a specific XML format. Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<dialog css="rheuma, nofoot, nonavigation"
        header="Rheumatic Disease Consultation"
        type="rheuma">
    <questionnaire>
        <question type="zc" title="Dear user..." send-button="true">
            <answer title="The following ..." />
        </question>
        <question answer-type="num" title="How old are you?" send-button="true">
            <answer title="years" />
        </question>
        <question answer-type="oc" title="What is your sex?" send-button="true">
            <answer title="male" />
            <answer title="female" />
        </question>
        <question type="image" answer-type="mc" image="human.png"
                  title="Please mark ..." send-button="true">
            <answer title="Head" shape="rect" coords="70,2,129,70" />
            <answer title="Left Shoulder" shape="rect" coords="42,76,85,105" />
        </question>
    </questionnaire>
</dialog>
```

In the dialog-tag, the basic dialog properties are set, as e.g. the CSS to be used or the dialog type, or dialog title. Then, a dialog basically consists of questions and answers, which can be further grouped (structured) by questionnaires (as in the example).

A complete listing of all elements and properties can be found in Appendix A

For automatically running the so specified dialog, the XML specification simply needs to be put in `src/main/resources/specs/prototypes` and is then automatically added to the ControlCenter Servlet, which provides an overview of all dialogs currently existing in the ProKEt project (in the respective specs folders).

2.6. ...create a prototype with d3web KB specification

This HowTo describes, how a dialog with knowledge base integration is created assuming that existing renderers / css styles / StringTemplate files are used.
(→ adapting styles, StringTemplate templates, and renderers: see 2.7, 2.8, and 2.9)

Therefore, two specification files are needed:

- The d3web knowledge base
  specifies the knowledge, i.e., basic dialog objects as questionnaires, questions, answers, additional descriptions etc
- The 'minimal XML specification' file
  This file contains all properties that are not (easily or reasonably) mappable by the d3web knowledge base. For example, it can specify whether the dialog should be extended by a login mechanism or not. An example:

```
<?xml version="1.0" encoding="UTF-8"?>
<dialog
    header="EuraHS"
    dtype="singleform"
    dstrategy="nextuaquestion"
    css="med2cols, nonavigation"
    dialogcolumns="1"
    questionnairecolumns="3"
    login = "no">
    <data kb="HerniaApr2011.d3web" />
</dialog>
```

As the example shows, e.g., the dialog title (header), css, login-yes/-no, but also properties as whether the questionnaire should consist of several columns (here: 3) are defined. In the < *data* > tag, the d3web knowledge base is referenced, that should be used for this dialog.
A complete listing of all elements and properties can be found in Appendix B

For automatically running a dialog, specified by those two files, both files need to be put into `src/main/resources/specs/d3web`; then the corresponding d3web dialog is automatically added to the ControlCenter servlet.

2.7. ...change CSS style

This HowTo shows, how already existing (pure or d3web) prototypes can be adapted as to use different CSS files or even single commands for specific dialog elements.

- Use a completely new CSS definition file
  1. Create new CSS file from scratch, specify the desired elements (listing of most often used base elements in Appendix E)
  2. Alternatively built upon existing CSS file(s).
     To avoid accidentally overwriting important base CSS files needed for the standard to-show-prototypes, please best copy & paste an existing CSS file that provides most or at least some of the desired styling, adjust it accordingly and name it properly.
     Those critical CSS files are listed in Appendix F.
  3. Include reference to the CSS file into XML specification by using the css='<cssfilename>' command within the <dialog> tag, both for the pure XML and minmal d3web XML specification
  4. Save the file and refresh the Servlet—either ControlCenter or single Dialog Servlet—on the Tomcat server.
- Add CSS commands to single dialog elements (currently only for pure prototypes).
  1. Search the specification of the element to be styled in the XML prototype specification file, and add the CSS command similar to adding the reference to a CSS file.
     An example: <question type='oc' columns='3' ... *css='border: none;'* ... >
  2. Save the file and refresh the Servlet—either ControlCenter or single Dialog Servlet—on the Tomcat server.

2.8. ...change HTML template(s)

TODO

2.9. ...write own/tailored renderers

TODO

2.10. ...add additional widgets

TODO

This HowTo explains, how additional widgets can be added to the system. Examples

are specific Popups, or Grids (for summarizing information). The mechanism is explained at the example of adding a Grid-Widget, that inserts grids that summarize specific information into the dialog.

- The knowledge base needs to be extended as to contain the additional information.
  In this case, the description is modified to contain a reference to the grid name, e.g. question.description = GRID_grid1, where GRID is the 'markup' for grid and 'grid1' the name of the grid.
- Alternatively, the additional information needs to be added to the minimal XML and the D3webXMLParser is to be adapted correspondingly.
- In the Question-Renderer we need to ask, whether the current question is marked as a GRID question.
- If so, we need to call the sub-Renderer (GridD3webRenderer?!) here.
- This renderer should fill a custom GridTemplate (Grid.st ?!) with the appropriate content.
- HERNIA: Maybe in OUR case we should have a super-Renderer GridD3webRenderer and then create one Renderer for each grid (e.g. Grid1HerniaRenderer... ?!) –¿ look through the grids in what point they vary etc...

2.11. ...update a d3web dialog due to KB update

To update a dialog due to an update of the KB...:

1. Copy the new knowledge base file into `src/main/resources/specs/d3web`
2. Check that knowledge base naming and KB reference in the minimal XML are equal, and adapt if necessary
3. Compile d3web-ProKEt (maven 'clean package install' with update snapshots)
4. Refresh target folder
5. Re-run ControlCenter, or only D3webDialog servlet, (or alternative, tailored servlet) on server

2.12. ...create own/tailored d3web dialog servlets

The default development servlet for d3web dialogs is `D3webDialog.java`. This provides all base functionality for running d3web dialogs, and for running them depending on the given `src` parameter in the query string. Thus, there are two ways for running d3web dialog servlets:

- `http://localhost:8080/d3web-ProKEt/ControlCenter`
  Via the ControlCenter, this lists all specified dialogs from the `specs/d3web` folder and the desired one just needs to be chosen from the d3web Dialogs sub-menu
- `http://localhost:8080/d3web-ProKEt/D3webDialog?src=Mediastinitis`
  directly opening the specific link in the browser, for a dialog specification named `Mediastinitis.xml`

(note: the localhost-link part might vary depending on the local tomcat installation/configuration)

However, it might be necessary or desired to create specific servlets, e.g., if special/adapted functionality is required only for a specific project.

In the following, we assume that a tailored servlet `MyProject` should be created and set up for running with the ProKEt base system.

1. Copy `D3webDialog.java` to `MyProject.java`
2. Adapt the servlet if necessary (e.g., only reference the one required XML specification file, for example, see the `Mediastinitis.java` tailored Servlet)
3. Add appropriate entries to the `web.xml == deployment descriptor` for the servlet class and URL mapping
4. Adapt `index.jsp` so it forwards to the correct servlet, here MyProject (instead of ControlCenter)
5. Compile d3web-ProKEt (maven compile 'clean package install', with 'update snapshots') and refresh target folder
6. `Rename` resulting snapshot to MyProject.war
7. Deploy MyProject.war on tomcat server
   - e.g. wms server 132.187.15.128, root, w3ltfrieden
   - stop tomcat, copy .war into webapps folder, restart tomcat
   - call url http://wmswiki.informatik.uni-wuerzburg.de/MyPro

2.13. ...global settings and user settings

There are two types of settings needed when running a dialog-servlet: *global* and user specific settings. They need to be distinguished, as there can be settings, that don't vary for different users (e.g., the knowledge base) and settings, that have to vary from user to user (e.g., the d3webSession). This is due to the fact, that the dialog app as a whole is running on one server, yet for every user a different dialog instance is created, to make it possible that different users can access and use the app from different locations at the same time.

- Global Settings:
  Are defined in the class `de.d3web.proket.utils.GlobalSettings.java`.
  There exists only one instance of this class (Singleton pattern); all settings defined here, are required and **don't change in different running dialog instances**
  Examples: knowledge base, basic system paths, the global persistence base folder, non-varying dialog properties (dialog type...)

- User Specific Settings:
  Are defined in the servlet classes and stored in the `HttpSession` of each user.
  Settings defined that way **vary from user to user** in the different running dialog instances.
  Examples: d3webSession, the latest loaded file/case (lastLoaded), needed when saving a case as to know whether the current user has the right to overwrite the already existing file (which he has when he loaded it previously), login-relevant data

2.14. ...make (JS) module insertion dynamic

Goal of this HowTo is, to enable a (JS-based) module (e.g., login mechanism) dynamically. That is, in the specification XML of the d3web-dialog something like $<$ *module* $>=<$ *value* $>$ should be provided and depending on value, the module should be added to the dialog or not. This HowTo explains the mechanisms on the example of the **login mechanism**.

The login mechanism consists of the following parts:

- `login.js and encrypt.js`
  Two JS files, that contain the definition and logic of the login dialog (which in this case is a JQuery UI dialog widget).

- `LoginDialog.st`
  A StringTemplate file, that defines the $<$ *div* $>$ element containing the dialog widget's elements (as input fields etc.)

To provide the possibility of inserting this module dependent on the specification in the d3web-dialog XML file, the following steps have to be taken:

- Adapt `D3webXMLParser.java`
  The parser needs to provide an adequate method to read and return the value from the XML spec properly (in this case: `getLogin`, a method returning a boolean TRUE if the login value is yes, FALSE, if the login value is no)

- Adapt the D3webDialog StringTemplate definition `D3webDialog.st`
  Here, we need to insert a dynamic StringTemplate part

```
$if(login)$
    $LoginDialog()$
$endif$
```

This basically means: if (somewhere in the rendering workflow) the login-StringTemplate variable is given a value, then the sub-StringTemplate `LoginDialog` is inserted. Otherwise it is just ignored.

- Adapt the `JSCodeContainer.java`
  In the JSCodeContainer we need a variable login (default: false), dependent on which (in `generateOutput()`), the desired JS-files are included or not:

```
if (login) {
    ownBibs.add("login.js");
    ownBibs.add("encrypt.js");
}
```

  Also, we need a method that sets the variable value, so it can be modified from öutside"(e.g., from the Renderers):

```
public void enableLogin() {
    login = true;
}
```

- Adapt the Renderer
  The appropriate Renderer—in our case: `D3webRenderer`—or alternatively a sub-Renderer needs to be adapted to set the StringTemplate variable to the required value. Also, the JS ContainerCollection needs to be told that it should include the login.js and encrypt.js.
  In the case of the login mechanism, in D3webRenderer we read the value from the D3webXMLParser, and if it returns "true"(meaning, login is to be inserted), we set the variable for the StringTemplate and notify the JSCodeContainer:

```
if (D3webConnector.getInstance().getD3webParser().getLogin()) {

    //set the attribute for the StringTemplate to include LoginDialog.st
    st.setAttribute("login", "true");

    // tell JSCodeContainer to set login att to true so JS is included
    cc.js.enableLogin();
}
```

Now it is possible to set the property "login = yesïn the d3web-dialog XML (see snippet below) and the JS based login mechanism is added to the dialog.

```
<?xml version="1.0" encoding="UTF-8"?>
<dialog
    header="EuraHS"
    css="med2cols, nonavigation"
    ...further properties...
    login = "no"
>
<data kb="HerniaApr2011.d3web" />
</dialog>
```

2.15. ...create skeletal Prototype (widget) Structures with StringTemplate

2.16. ...use predefined JS/JQuery(UI) libs

- put javascript stuff of lib into webapp/js/SUBFOLDER (for jquerystuff e.g. /jquery)
- link library to where it is needed:
  - for ControlCenter modify controlcenter.st
  - for Dialog to use it, modify JSCodeContainer (for libs that are common to all potential dialogs to come) or put it directly into a Renderer (see example in D3webRenderer)
- if css/images are needed, put them into webapp/css/SUBFOLDER(optionally)
- link to css where it is needed:
  - as before also in CSSCodeContainer or directly in Renderer class

Appendix

A. XML Spec. Pure Prototyping

B. XML Spec. d3web Prototyping

C. Basic Renderers in ProKEt

D. Basic Dialog Types in ProKEt

E. Common CSS classes

Overview/Listing of basic CSS classes used for most prototypes...

F. Basic CSS files (DO NOT change)

Listing of basic CSS files that are required to run the default/standard dialog types. Those are **NOT TO BE CHANGED**. If a dialog is to built upon existing styles, please copy & paste (and rename) those files, but do NOT overwrite.

G. Basic StringTemplate files (NOT to be changed!)

Listing of basic StringTemplate files that are required to run the default/standard dialog types. Those are **NOT TO BE CHANGED**. If a dialog is to built upon existing styles, please copy & paste (and rename) those files, but do NOT overwrite.

H. Troubleshooting